正课:

1. ES5

2. ES6

1. ES5

保护对象

保护对象的属性:

保护对象的结构: 3个级别:

1. 防扩展: 禁止添加新属性:

Object.preventExtensions(obj)

阻止对obj对象添加任何扩展属性

原理: 每个对象内部都有一个隐藏的extensible的属性，默认值为true。

preventExtensions(obj)其实就是将这个内部的隐藏属性改为false！

2. 密封: 在兼具防扩展同时，进一步禁止删除现有属性

Object.seal(obj)

2件事: 1. 禁止添加新属性

obj的extensible=false

2. 禁止删除现有属性:

自动将所有属性的configurable都改为false！从此defineProperties中不需要再重复写configurable:false!

其实，今后绝大多数对象，都要密封！

强调: 虽然密封禁止修改结构，但是属性值随便改！

3. 冻结: 在兼具密封的基础上，进一步禁止修改属性的值！

Object.freeze(obj)

3件事: 1. 禁止添加新属性

obj的extensible=false

2. 禁止删除现有属性:

自动将所有属性的configurable都改为false！

3. 自动将所有属性的writable都改为false。

Object.create()函数: 没有构造函数的情况下，也想创建一个子对象。

3件事: 1. 创建一个新对象

2. 继承一个父对象

3. 为新对象添加自有属性

var 子对象=Object.create(父对象,{

//defineProperties

属性名: {

value: 属性值,

writable: true/false,

enumerable:true/false,

configurable:true/false

},

... ...

})

call, apply, bind: 替换this

何时: 只要函数执行时，内部的this不是我们想要的，就可用这三个函数，将不想要的this换成任意一个想要的对象。

如何:

1. call, apply: 在这一次调用函数时，临时替换一次this为任意指定的对象！

要调用的函数.call(替换this的对象, ...)

强调: 实参值列表必须从第二个位置开始。因为第一个实参的位置让给了替换this的对象！

原理:



call的更大的作用: 可让任何对象去调用原本没有任何关系的一个函数。——抢！

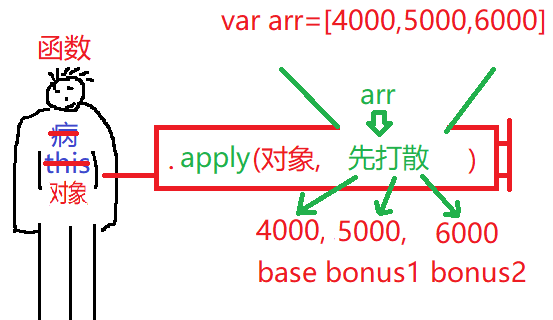
apply vs call:

apply和call用法几乎完全一样。只不过，要求所有实参值都要放在一个数组中整体传入。

执行过程:

1. 先用第一个参数值对象替换函数中的不想要的this。

2. 打散数组为多个单独的元素值，分别传给对应的每个形参变量。



总结: 通常情况，要替换函数中的this，用call就够了！只有实参值列表是放在一个数组中的时候，才需要apply，先打散数组，再传入函数。

问题: 只能一次性临时替换this。如果反复调用，反复替换，代码会很繁琐！

2. bind: 不调用函数，而是基于原函数，创建一个新函数副本。并永久替换新函数中的this为指定的对象。

何时: 今后，如果一个函数需要反复调用，又反复替换其中的this时，都用bind()来创建副本，并永久绑定this

如何:

var 新函数=旧函数.bind(

替换this的对象,

要绑定的实参值,

... ...

)

结果: 1. 新函数的功能和旧函数是完全一样的。

2. 但是新函数中的this和开头部分形参，已被永久替换为固定的对象和固定的参数值。

3. 今后调用新函数时，只要传入可能变化的剩余几个实参值即可！

数组函数:

1. 在数组中查找一个元素的位置: indexOf

数组的indexOf和字符串的indexOf完全一样

var i=arr.indexOf("元素", fromi)

在arr数组内，从fromi位置开始，找下一个和指定"元素"相同的元素所在的位置i

返回值: 如果找到，返回i

如果没找到，返回-1

2. 判断:

1. 判断数组中是否所有元素都符合要求

var bool=arr.every(function(value, i, arr){

//value: 当前元素值

//i: 当前下标位置

//arr: 当前数组对象

return 判断条件

})

原理: every会自动遍历arr中每个元素。每遍历一个元素，就自动调用一次回调函数function。每次调用function时，自动传入当前正在遍历的元素值和下标位置，以及当前数组对象。每次调用function，都会用函数内的条件检查当前元素是否符合要求。只有当前元素符合要求，才继续遍历下一个元素。只要碰上一个不符合要求的元素，就立刻退出遍历。并返回false。意思是不是所有元素都满足要求。如果遍历结束，都还没有退出，说明整个数组中所有元素都符合要求。整个every()，就返回true。

2. 判断数组中是否包含符合要求的元素

var bool=arr.some(function(value,i,arr){

return 判断条件

})

用法和every完全相同

只不过执行时，只要碰到一个元素判断为true，就不再遍历，直接返回true。只有遍历结束，还没有找到判断为true的元素，才返回false！

3. 遍历:

1. 普通遍历: 代替for循环

for(var i=0;i<arr.length;i++){

//i 当前位置

//arr[i] 当前元素值

}

arr.forEach(function(value, i, arr){

//对当前元素执行的操作！

})

forEach中自动遍历arr中每个元素，每遍历一个元素，就在这个元素上自动调用一次回调函数function。自动传入当前元素值给value, 当前位置给i, 当前数组对象给arr。

2. 遍历并复制原数组为新数组:

var 新数组=原数组.map(

//var newArr=[]; //自动

//for(var i=0;i<arr.length;i++){ //自动

// arr[i] i this

// ↓ ↓ ↓

function(value, i, arr){

return 要放入新数组中的元素值

}

//}

//return newArr; //自动

)

原理:

1. map内先自动创建一个新数组

2. 然后才是带着回调函数一起遍历原数组。自动将回调函数在每个元素上执行一次。将回调函数return的结果，放到新数组中相同的位置上

3. 遍历结束，返回新数组

总结:

1. 仅单纯代替for循环遍历数组，首选forEach。

2. 除非想保护原数组，返回新数组时，采用map.

4. 过滤和汇总:

1. 过滤: 复制出原数组中符合条件的元素组成新数组返回。原数组保持不变！

var 新数组=原数组.filter(  
 function(value, i, arr){

return 判断条件

}

)

原理:

1. 先自动创建一个空数组

2. 再自动遍历原数组中每个元素。每遍历一个元素，就自动调用一次回调函数。如果当前元素经过回调函数验证，返回true，则加入新数组中。如果当前元素经过回调函数验证，返回false，则不加入新数组。

3. 遍历结束，返回新数组！

2. 汇总: 对数组中的元素内容进行统计，得出最终结论。

var sum=arr.reduce(

function(prev, elem,i,arr){

//prev 保存截止到当前元素的临时汇总值。

return 将当前元素elem，累加到prev中，得到的新的汇总值。

},

起始值

)

原理: 看视频。

其实，reduce可以很复杂。

作业：

1. 如何用ES5实现两种类型间的继承 ——手写

2. 如何判断一个对象是不是数组。共有几种方法！